

THE PATENTS ACT, 1970

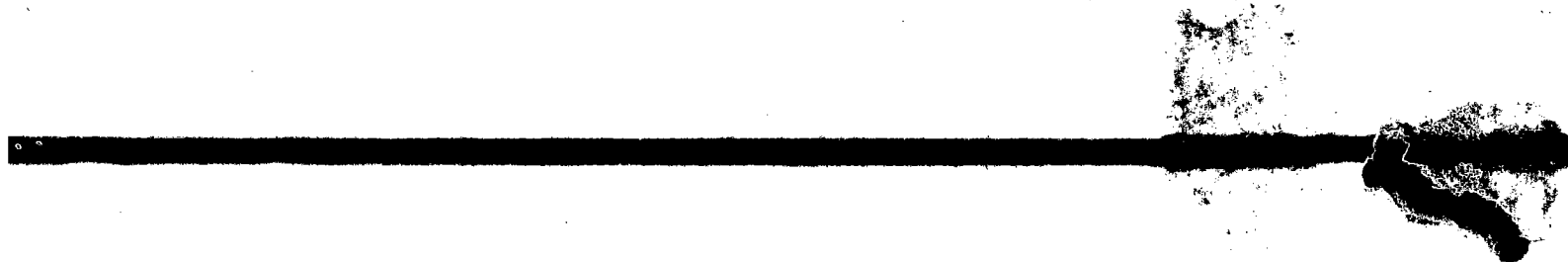
IT IS HEREBY CERTIFIED THAT, the annex is a true copy of Application and Provisional specification filed on 17.07.2001 in respect of Patent Application No. 689/MUM/2001 of Tata Consultancy Services (a Division of Tata Sons Ltd.) of Bombay House, Sir Homi Mody Street, Mumbai-400 023, Maharashtra, India an Indian Company .

This certificate is issued under the powers vested on me under Section 147(1) of the Patents Act, 1970.

..... Dated this 3rd day of April 2003


(N.K.GARG)

ASST CONTROLLER OF PATENTS & DESIGNS



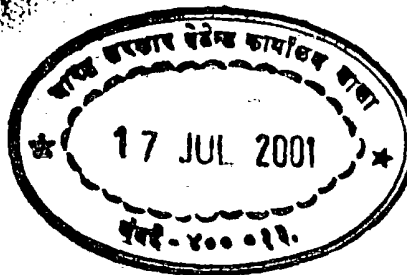
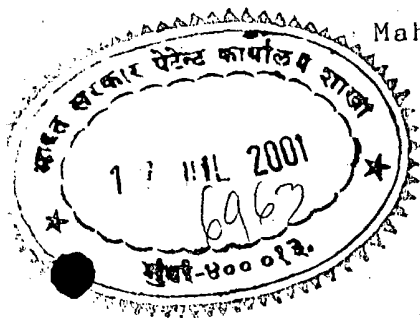
THE PATENTS ACT, 1970

(39 OF 1970)

APPLICATION FOR GRANT OF A PATENT

(See sections 5(2), 7, 54 and 135 and rule 33A)

1. KWe, TATA CONSULTANCY SERVICES (a Division of TATA SONS LTD.)
of Bombay House, Sir Homi Mody Street, Mumbai 400 023,
Maharashtra, India, an Indian Company



2. hereby declare :-

(a) that ~~xxx~~/we are in possession is an invention titled A METHOD AND APPARATUS
FOR VERSIONING AND CONFIGURATION MANAGEMENT OF MODELS

(b) that the Provisional/~~Complete~~ specification relating to this invention is filed with this
application

(c) that there is no lawful ground of objection to the grant of a patent to ~~xxx~~/us.

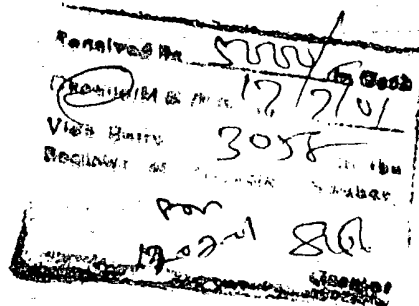
3. further declare that the inventor(s) for the said invention~~xx~~are :

- a) SREEDHAR SANNAREDDY REDDY of Tata Consultancy Services, Hadapsar
Industrial Estate, Pune 411 013, Maharashtra, India, an Indian National;
and
b) ARUN GAJANAN BAHULLAR of Tata Consultancy Services, Hadapsar
Industrial Estate, Pune 411 013, Maharashtra, india, an Indian National

689/mum/2001

689/मुंबई 2001

17 JUL 2001



ORIGINAL

P₁
cash
17/7/01

Electronic

- (a) [Country]
(b) [Appln.No.]
(c) [Date of Appln.]
(d) [Applicant in Convention Country]
(e) [Title]

N.A.

5. I/We state that the said invention is an improvement in or modification of the invention, the particulars of which are as follows and of which I/We are the applicant/patentee :

- (a) Application No.
(b) Date of application

N.A.

6. I/We state that the application is divided out of my/our application, the particulars of which are given below and pray that this application deemed to have been filed on _____ date under section 16 of the Act.

- (a) Application No.

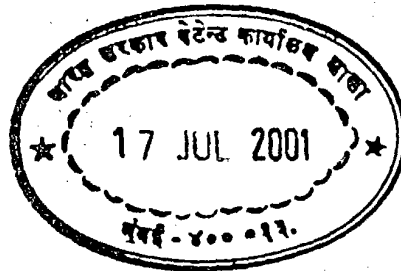
- (b) Date of filing of provisional specification :

N.A.

and date of filing of complete specification :

7. That I am/We are the assignee or legal representative of the true and first inventors.

8. That my/our address for service in India is as follows : R.K. DEWAN & COMPANY, Trade Marks & Patents Attorneys, 78, Podar Chambers, S.A.Brelvi Road, Fort, Mumbai 400 001, Maharashtra, India, Tel. (91) 22-2661662/2663002, Telefacsimile number (91) 22-2650159, Email>rkdewan@vsnl.com.



9. Following declaration was given by the inventor(s) or applicant(s) in the convention country :

~~I/We~~ the true and first inventors for this invention ~~or the applicant(s) in the convention country~~ declare that the applicant(s) herein is/are my/our assignee or legal representative :

Sreedhar

(SREEDHAR SANNAREDDY REDDY) (ARUN GAJANAN
BHULLAR)

Arun

10. That to the best of ~~my~~/our knowledge, information and belief the fact and matters stated herein are correct and that there is no lawful ground of objection to the grant of patent to me/us on this application



(a) Provisional specification (3 copies)

(b) Drawings (3 Copies)

(c) ~~Priority documents~~

(d) ~~Statement of Undertaking on FORM 3~~
copy of General

(e) Power of authority

(f)

(g)

(h)

(i) Fee Rs. In cash/cheque/bank draft bearing No.

date

on

Bank

x/We request that a patent may be granted to me/us for the said invention.

Dated this 16th day of July 2001

for TATA CONSULTANCY SERVICES

Signature :

Name : (HEMANT DHANSUKHLAL, VAKIL)

To :

The Controller of Patents

The Patent Office

at MUMBAI



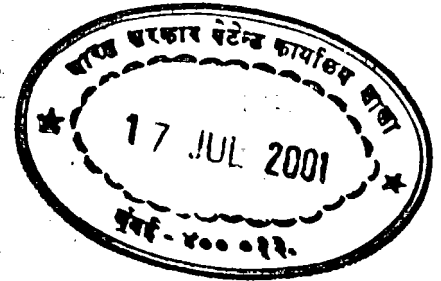
FORM-2

THE PATENT ACT, 1970

PROVISIONAL

Specification

SECTION - 10



A METHOD AND APPARATUS FOR VERSIONING AND
CONFIGURATION MANAGEMENT OF MODELS

TATA CONSULTANCY SERVICES,

(a Division of TATA SONS LIMITED),

of Bombay House, Sir Homi Mody Street, Mumbai 400 023,

Maharashtra, India,

an Indian Company

THE FOLLOWING SPECIFICATION DESCRIBES

THE NATURE OF THIS INVENTION :-

689 | मुंबई | 2001
MUM

17 JUL 2001

ORIGINAL

This invention relates to a method and apparatus for versioning and configuration management of models.

The software industry is gradually moving to a new paradigm of development, where modeling is moving from being just an analysis/design aid to being a more pervasive end-to-end system development tool. This new paradigm views modeling and coding on a continuum, with more and more things being pushed from the domain of coding to the domain of modeling. This shift is being facilitated by the emergence of various modeling standards like Unified Modeling Language(UML), Meta Object Facility(MOF), Extended Markup Language(XML), etc., with active ongoing efforts to make them ever more semantically richer.

When modeling takes over the space of coding, it will have to contend with the problems of size, change and variation just as coding does now. To manage the complexity of size, information systems are partitioned into modules or components with well-defined interfaces. As the information system evolves, its models also evolve. A successful information system typically has many variants specific to various delivery platforms, geographical regions, and multiple development streams. Versioning is a natural consequence of evolution and variation. Without adequate tool support, the problem of versioning and configuration management can quickly get out of control.

2. State of the art

In the coding world, there exist several versioning and configuration management tools like SCCS, CVS, VSS, etc, that help in managing these problems. However, these tools are not suitable for the modeling world as they are based on flat files. They treat files as lines of text and do not understand the semantic entities and their relationships contained in the files.

On the other hand, in the modeling world, there exist some repository systems that provide support for versioning of models to some extent. However, these do not go far enough as the versioning model supported by them largely mimics the file-system model, in that they treat the system as a set of objects that can be versioned, without paying much attention to the relationships between the objects. Users are responsible for assembling related objects into meaningful configurations. There is no support for addressing questions like:

Is the configuration complete?

Does it have all the required objects?

Are the objects compatible with each other?

Such a solution does not scale up to the demands of large software systems, as it is extremely difficult for users to manually assemble large numbers compatible objects into configurations that are complete.

This invention envisages a versioning and configuration management method and apparatus therefor that addresses the above issues by exploiting the relationships captured in the models.

The method and apparatus therefor is especially well suited to generic, extensible model-repository systems that are meta-model driven.

A modeling framework (mappable to MOF) for generic, extensible repository systems is illustrated in Figure 1 of the accompanying drawings.

The meta model shown in figure 1 is the base model. It is the schema for modeling meta models. The meta meta model is capable of describing itself, i.e., it can model itself. It is the root of the instantiation hierarchy. The meta meta model consists of objects, associations and properties seen in Table 1 shown in Figure 5 of the accompanying drawings.

In this specification a meta model is an instance of the meta meta model. It consists of Meta Objects with associated Meta Properties, and Meta Associations defined between Meta Objects. A meta model defines the structure and semantics for the information system model (e.g.: UML meta model).

Information system model or User model

The information system model, also referred to as the user model, is an instance of a meta model. It captures the description of the

information system from various points of view as specified by the meta model (e.g.: UML model of a banking system).

The above modeling framework is abstract enough to be able to support modeling method and apparatus therefors like UML, ER, and the like.

The versioning and Configuration Management method and apparatus envisaged in accordance with this invention can be described as follows:

A versioning and configuration management system for managing models that are compliant with the above modeling framework is shown in Figure 2 of the accompanying drawings. In the model, 'object' refers to an element instance of either the user or the meta model. The model shown in the figure 2 can come as a pre-defined meta-model in the repository.

Component

Component is a construct for partitioning an information system model. Each module of an information system can be mapped to one or more components. The model for each module can then be placed in its components. A component is a container for model elements like objects, properties and associations. An object on its creation is placed in a component and carries the identity of the component. Versioning of models is done at the level of a component. A component is realized by its versions. The first version of a component is created when the component is created. All other

versions are derived, directly or indirectly, from the first version. Versions of compatible components are assembled, within a configuration, to represent the complete model of an information system.

An object belongs to a component and is versioned with the component. Thus, an object gets the version number of its container component, but two versions of an object will always have the same ObjectId. When an object is contained in a component version, its properties and the associations owned by it are also contained in that component version.

Component is a typeless construct in that an instance of any meta object can be placed in a component. Versioning is implemented at the component level and not at the object level because an object by itself is usually too fine-grained an entity for versioning. An object in a model is always seen in the context of its associated objects.

Components provide a mechanism for grouping objects for the purpose of versioning; grouping can be done at any required level of granularity.

Associations can either be intra-component or inter-component. An association between objects belonging to different component versions is an inter-component association. An inter-component association establishes a relationship between two component versions and it belongs to the component version to which the owner object of the association belongs. Inter-component associations can exist only in the context of a configuration.

A component version guarantees change isolation. Changes made to the objects in a component version are not visible in other versions of the component or other components. Conceptually, a new version of a component contains copies of the objects of the version from which it is derived. A component enables concurrent development of different parts of the model. A component version can be shared between various configurations.

Figure 3 of the accompanying drawings shows two configurations of the method and apparatus of this invention in its operative configuration. (Banking Application Analysis and Design), versions of components (Business Partner and Accounts) and objects (Customer, Account, Balance, etc.). Two versions of Account object can be seen. The first version in the Analysis Configuration has associations with two Attributes: AccId and Balance. The second version of the Account object in the Design Configuration has associations with an Operation (Withdraw) and a Table (AccTbl) in addition to having associations with the Attributes.

The version management facility provides support for tracking the history of changes to the components, branch versioning, selection of different versions of different components to form a meaningful configuration, and 'diff and merge' of models contained in different versions of components.

Configuration

A configuration is a container for assembling different versions of components. A configuration represents the notion of 'a complete unit

of work'; it can be seen as a product; as an input to a set of code generation tools, or as an input to a 'make' utility. A configuration is a complete set of compatible components and provides the context for creating relationships between various components via the inter-component associations. Only one version of a given component can belong to a configuration. A configuration can contain other configurations. Since a configuration is defined as an independent and complete entity, a container configuration can 'use' or 'depend on' the contained configuration but not vice versa.

A configuration is realized through its versions. Unlike component versions, two versions of a configuration can overlap, i.e. they can share component versions. For example, in figure 3, the component version 'Business Partner Module V1.0' is shared by the analysis and design configurations. The versioning of configurations is provided for the purpose of tracking the version history only. Thus, configuration versions are designed to support sharing, while component versions are designed to support change isolation.

Completeness of a Configuration

The "ownership" property of associations helps define and enforce the notion of completeness of a configuration with regard to the component versions it contains. The inter-component associations in a configuration establish 'consumer-supplier' relations between component versions. A component version that owns an inter-component association depends on the associated component version. The owner object (and hence its container component) of such an association is deemed 'incomplete' without the associated object (and

hence its container component). Thus, a configuration is complete only if it contains a complete set of related component versions that satisfy all the 'owned' inter-component associations within the context of the configuration. This notion of completeness is enforced for all configurations ensuring that all the 'consumer-supplier' relations are satisfied when components are assembled into a configuration.

Figure 4 of the accompanying drawings shows examples of complete and incomplete configurations. The 'owned' association 'Uses' of Class Customer is not satisfied in configuration A, hence making it incomplete.

Compatibility of components

Inter-component associations provide a mechanism to establish and enforce compatibility semantics between two component versions. A version of an object can replace another version of the object in an association. Hence, two versions of an object are deemed compatible. An object is shared if its component version is shared in more than one configuration. Such a shared object must appear the same, with respect to its properties and owned-associations in all the sharing configurations. Changes made to a shared object within a configuration must be reflectable in all the sharing configurations. Specifically, addition and deletion of owned-associations to a shared object should be possible in all the sharing configurations. The configuration A in figure 4 shows an invalid sharing condition for class 'Customer' in the shared component 'Business Partner V1.0'. A set of component versions are deemed compatible if they can be put

together in a configuration without causing any sharing condition violation with respect to inter-component associations. These notions of sharing and compatibility are always enforced by the system.

The notion of component compatibility is based on the premise that two versions of an object are compatible with each other. This notion of compatibility is designed to allow maximum possible flexibility in component composition by enforcing only the minimum required constraints.

Diff and Merge

Quite often, a component needs to be developed concurrently by independent teams in a non-interfering way. This can be achieved by deriving a branch version of the component. At the end of the development effort, the branch version of the model needs to be merged back into the main stream.

A facility needs to be provided to compare ('Diff') and merge models contained in two component versions or two configurations. The 'Diff' operation can be done at the level of configurations, component versions, or objects.

Two versions of an object are compared based on their Object Ids and in terms of their property values and associations. Often, comparison of two objects by themselves is not of much value unless it is done in the context of its associated objects. Such a context is also necessary while merging versions of an object. In general, in a given model, some objects play the role of primary objects and others play the role

of companion objects. For example, in the OO model, Class is a primary object and Data Member is a companion object. Two Data Members should only be compared in the context of the Classes to which they belong. Such context sensitive comparison and merge operations can be performed by specifying object-association graphs or patterns. An example pattern for comparing classes in two components could be: 'Class-HasData-DataMember', 'Class-HasMethod-Method'. It is also possible to specify the list of properties to be compared. In addition to providing a context, a pattern also limits the scope of comparison to models of interest.

Advantages of the method and apparatus envisaged in accordance with this invention include the following among others:

1. Association 'ownership' property can model notions like 'client-supplier relationship', 'depends on', 'can not exist without', etc., and provides the basis for relationship driven configuration management.
2. Component provides a better handle over the granularity of the unit of versioning. 'Object' is too fine-grained an entity for version tracking. A component allows related objects to be put together in a container and treat that as the unit of versioning. This solution scales up better than object versioning.
3. Component versioning provides change isolation. Configurations provide controlled sharing, by enforcing sharing semantics. Controlled sharing avoids the need for frequent merges, which are error prone. If required, traditional check-out/check-in model of versioning can be easily simulated using components and configurations.

4. The properties of 'configuration completeness' and 'component compatibility' greatly help in composing configurations correctly.
5. Pattern-based 'diff/merge' feature facilitates in conducting intelligent, structure-driven comparison and merge operations. These operations can be performed at various levels: object, component version or configuration.

Although the invention has been described in terms of particular embodiments and applications, one of ordinary skill in the art, in light of this teaching, can generate additional embodiments and modifications without departing from the spirit of or exceeding the scope of the invention. Accordingly, it is to be understood that the drawings and descriptions herein are proffered by way of example to facilitate comprehension of the invention and should not be construed to limit the scope thereof.

Dated this 16th day of July 2001



Mohan Dewan

Of R. K. Dewan & Co.,
Applicants' Patent Attorney

TATA CONSULTANCY SERVICES
NAME : (TATA SONS LIMITED)

NO. : /MUM/01

NO. OF SHEETS : 5

SHEET NO. : 1

PROVISIONAL SPECIFICATION

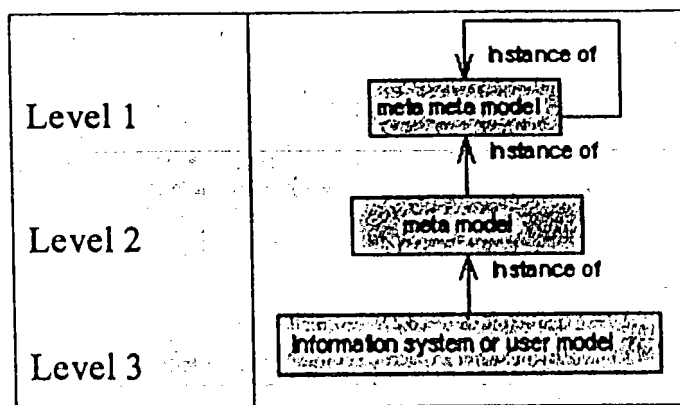
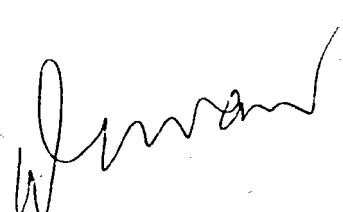


FIGURE - 1



DR. MOHAN DEWAN
APPLICANT'S PATENT ATTORNEY

17 JUL 2001

PROVISIONAL SPECIFICATION

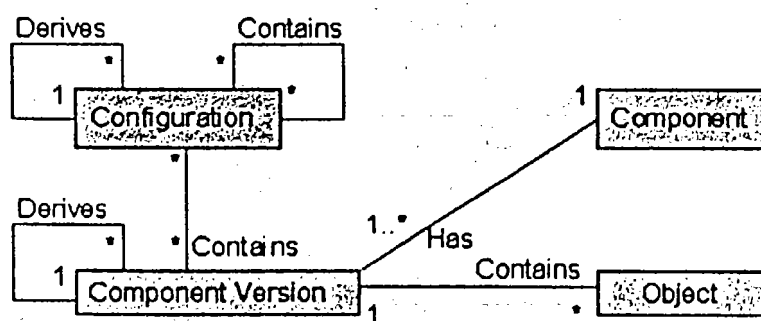


FIGURE - 2

DR. MOHAN DEWAN
APPLICANT'S PATENT ATTORNEY

17 JUL 2001

PROVISIONAL SPECIFICATION

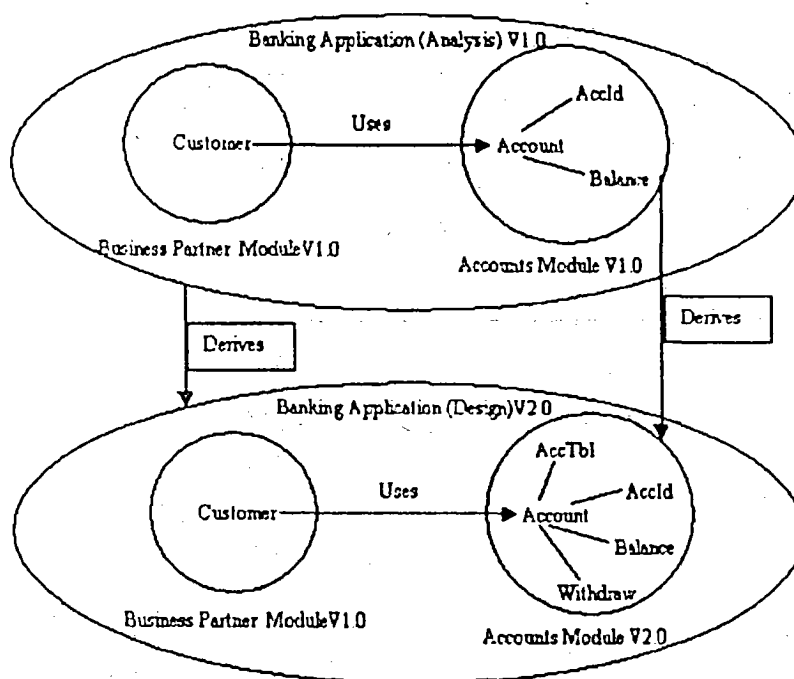


FIGURE - 3

DR. MOHAN DEWAN
APPLICANT'S PATENT ATTORNEY

17 JUL 2001

PROVISIONAL SPECIFICATION

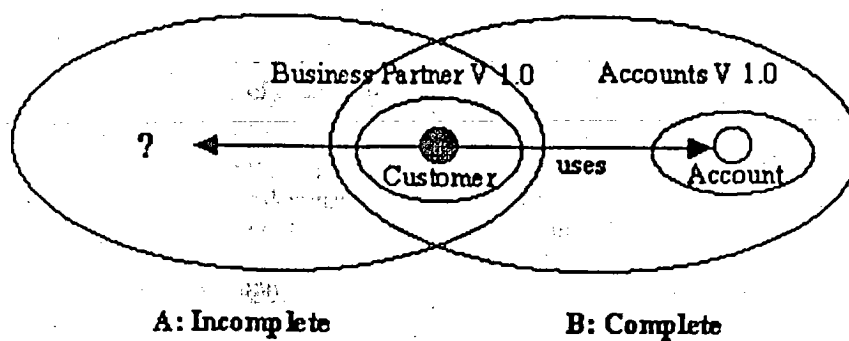


FIGURE - 4

17 JUL 2001

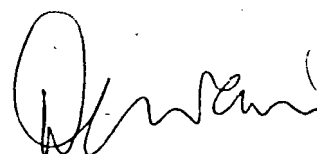
[Signature]
DR. MOHAN DEWAN
APPLICANT'S PATENT ATTORNEY

PROVISIONAL SPECIFICATION

Objects in Meta Meta Model	Properties of Object
MetaObject	Name, Description, AbstractConcrete
MetaProperty	DataType = Char, Number, Binary Size = Size of Char String and Number
MetaAssociation	Forward and Reverse Name Source and Destination Optionality = Association is optional or mandatory for source or destination object Source and Destination Cardinality = One or Many Owner of the Association = Source or destination object is the owner of the association

Associations in Meta Meta Model	Properties of Association
MetaObject InheritsFrom MetaObject	A MetaObject inherits associations and properties from another MetaObject. Many to Many; Optional
MetaObject Has MetaProperty	MetaObject has MetaProperties. One to Many; Optional
MetaObject SourceOf MetaAssociation	MetaObject is source of a MetaAssociation. One to Many; Mandatory for MetaAssociation
MetaObject DestinationOf MetaAssociation	MetaObject is destination of a MetaAssociation. One to Many; Mandatory for MetaAssociation

FIGURE - 5



DR. MOHAN DEWAN
APPLICANT'S PATENT ATTORNEY

17 JUL 2009

9.

